CS 2770: Computer Vision

Grouping & Transformations

Prof. Adriana Kovashka
University of Pittsburgh
February 4, 2020
Plan for this lecture

• Group pixels into:
  – Edges: Extract gradients and threshold
  – Lines: Find which edge points are collinear or belong to another shape
  – Segments: Find which pixels form a consistent region, e.g. via clustering

• Transform pixels:
  – Find relationships between multiple views of the same world point
  – *Both parts rely on finding geometric relationships between pixels*
Edge detection

- **Goal**: map image from 2d array of pixels to a set of curves or line segments or contours.
- **Why?**

- **Main idea**: look for differences in intensity, i.e. find strong gradients, then post-process

Figure from J. Shotton et al., PAMI 2007

Adapted from K. Grauman
What causes an edge?

Reflectance change: appearance information, texture

Depth discontinuity: object boundary

Cast shadows

Adapted from K. Grauman
An edge is a place of rapid change in the image intensity function, which corresponds to the extrema of the derivative.
Now with a little noise...

• Consider a single row or column of the image
  – Plotting intensity as a function of position gives a signal

\[ f(x) \]

\[ \frac{d}{dx} f(x) \]

Where is the edge?

Source: S. Seitz
Without noise

• Consider a single row or column of the image
  – Plotting intensity as a function of position gives a signal

\[ f(x) \]

\[ \frac{d}{dx} f(x) \quad \text{Diff} = 1 \]

Where is the edge?

\[ = \frac{\Delta f(a)}{\Delta a} = \frac{f(a + h) - f(a)}{(a + h) - a} = \frac{f(a + h) - f(a)}{h} \]

With noise

- Consider a single row or column of the image
  – Plotting intensity as a function of position gives a signal

Where is the edge?

\[
\frac{d}{dx} f(x) = \frac{\Delta f(a)}{\Delta a} = \frac{f(a + h) - f(a)}{(a + h) - a} = \frac{f(a + h) - f(a)}{h}
\]
Solution: smooth first

To find edges, look for peaks in $\frac{d}{dx}(f * g)$.

Source: S. Seitz
Derivative theorem of convolution

• Differentiation is convolution, and convolution is associative:
  \[ \frac{d}{dx} (f \ast g) = f \ast \frac{d}{dx} g \]

• This saves us one operation:

![Image with edge](source: S. Seitz)

Derivative of Gaussian

Edge = max of derivative

Source: S. Seitz
Canny edge detector

• Filter image with derivative of Gaussian
• Find magnitude and orientation of gradient
• Threshold: Determine which local maxima from filter output are actually edges
• Non-maximum suppression:
  – Thin wide “ridges” down to single pixel width
• Linking and thresholding (hysteresis):
  – Define two thresholds: low and high
  – Use the high threshold to start edge curves and the low threshold to continue them

Adapted from K. Grauman, D. Lowe, L. Fei-Fei
Example

input image ("Lena")
Derivative of Gaussian filter

Source: L. Lazebnik
Compute Gradients

X-Derivative of Gaussian  Y-Derivative of Gaussian  Gradient Magnitude

Source: D. Hoiem
Thresholding

• Choose a threshold value $t$
• Set any pixels less than $t$ to 0 (off)
• Set any pixels greater than or equal to $t$ to 1 (on)

Source: K. Grauman
The Canny edge detector
	norm of the gradient (magnitude)

Source: K. Grauman
The Canny edge detector

Source: K. Grauman
Another example: Gradient magnitudes
Thresholding gradient with a lower threshold

Source: K. Grauman
Thresholding gradient with a higher threshold  

Source: K. Grauman
The Canny edge detector

How to turn these thick regions of the gradient into curves?

Source: K. Grauman
Non-maximum suppression

- Check if pixel is local maximum along gradient direction
- Compare to pixels immediately neighboring on both sides
  - i.e. compare $q$ to $p$ and $r$
- Requires checking interpolated “pixels” $p$ and $r$ (at non-integer locations, so no intensity information) – *bilinear interpolation*

Adapted from K. Grauman
Bilinear interpolation

\[ f(x, y) \approx \begin{bmatrix} 1 - x & x \end{bmatrix} \begin{bmatrix} f(0, 0) & f(0, 1) \\ f(1, 0) & f(1, 1) \end{bmatrix} \begin{bmatrix} 1 - y \\ y \end{bmatrix}. \]
Related: Line detection (fitting)

• Why fit lines?
  Many objects characterized by presence of straight lines

• Why aren’t we done just by running edge detection?
Difficulty of line fitting

- **Noise** in measured edge points, orientations:
  - e.g. edges not collinear where they should be
  - how to detect true underlying parameters?

- **Extra** edge points (clutter):
  - which points go with which line, if any?

- Only some parts of each line detected, and some parts are missing:
  - how to find a line that bridges missing evidence?

Adapted from Kristen Grauman
Least squares line fitting

- **Data:** \((x_1, y_1), \ldots, (x_n, y_n)\)
- **Line equation:** \(y_i = mx_i + b\)
- **Find** \((m, b)\) to minimize

\[
E = \sum_{i=1}^{n} (mx_i + b - y_i)^2
\]

where line you found tells you point is along y axis
where point really is along y axis
You want to find a single line that “explains” all of the points in your data, but data may be noisy!

\[
E = \sum_{i=1}^{n} \left( \begin{bmatrix} x_i & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - y_i \right)^2 = \left\| \begin{bmatrix} x_1 \\ 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - \begin{bmatrix} y_1 \\ y_n \end{bmatrix} \right\|^2 = \|Ap - y\|^2
\]

**Matlab:** \(p = A \backslash y;\)

Adapted from Svetlana Lazebnik
Outliers affect least squares fit
Outliers affect least squares fit
Dealing with outliers: Voting

- **Voting** is a general technique where we let the features vote for all models that are compatible with it.
  - Cycle through features, cast votes for model parameters.
  - Look for model parameters that receive *a lot of votes*.

- Noise & clutter features?
  - They will cast votes too, *but* typically their votes should be inconsistent with the majority of “good” features.

- Two common techniques
  - Hough transform
  - RANSAC

Adapted from Kristen Grauman
Finding lines in an image: Hough space

Connection between image (x,y) and Hough (m,b) spaces

$y = m_0 x + b_0$ • A line in the image corresponds to a point in Hough space
Finding lines in an image: Hough space

A line in the image corresponds to a point in Hough space.

What does a point \((x_0, y_0)\) in the image space map to?

- Answer: the solutions of \(b = -x_0m + y_0\)
- This is a line in Hough space
- Given a pair of points \((x, y)\), find all \((m, b)\) such that \(y = mx + b\)

Adapted from Steve Seitz
Finding lines in an image: Hough space

What are the line parameters for the line that contains both \((x_0, y_0)\) and \((x_1, y_1)\)?

- It is the intersection of the lines \(b = -x_0m + y_0\) and \(b = -x_1m + y_1\)
Finding lines in an image: Hough space

How can we use this to find the most likely parameters \((m,b)\) for the most prominent line in the image space?

- Let each edge point in image space vote for a set of possible parameters in Hough space.
- Accumulate votes in discrete set of bins; parameters with the most votes indicate line in image space.
Finding lines in an image: Hough space

Adapted from Silvio Savarese
Parameter space representation

- Problems with the (m,b) space:
  - Unbounded parameter domains
  - Vertical lines require infinite m

- Alternative: *polar representation*

\[ x \cos \theta + y \sin \theta = \rho \]

Each point \((x,y)\) will add a sinusoid in the \((\theta,\rho)\) parameter space.
Problems with the (m,b) space:

- Unbounded parameter domains
- Vertical lines require infinite m

Alternative: polar representation

Each point (x,y) will add a sinusoid in the (θ,ρ) parameter space
Algorithm outline: Hough transform

• Initialize accumulator H to all zeros

• For each edge point \((x, y)\) in the image
  
  For \(\theta = 0\) to \(180\)
  
  \[ \rho = x \cos \theta + y \sin \theta \]
  
  \[ H(\theta, \rho) = H(\theta, \rho) + 1 \]

  end

end

• Find the value(s) of \((\theta^*, \rho^*)\) where \(H(\theta, \rho)\) is a local maximum

• The detected line in the image is given by
  
  \[ \rho^* = x \cos \theta^* + y \sin \theta^* \]
Incorporating image gradients

• Recall: when we detect an edge point, we also know its gradient direction
• But this means that the line is uniquely determined!

• Modified Hough transform:

\[ \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \]

\[ \theta = \tan^{-1} \left( \frac{\partial f}{\partial y} / \frac{\partial f}{\partial x} \right) \]

For each edge point \((x,y)\) in the image

\[ \theta = \text{gradient orientation at } (x,y) \]

\[ \rho = x \cos \theta + y \sin \theta \]

\[ H(\theta, \rho) = H(\theta, \rho) + 1 \]

end
Hough transform example
Impact of noise on Hough

Image space edge coordinates

Votes
Impact of noise on Hough

Image space
edge coordinates

Votes

What difficulty does this present for an implementation?
Voting: practical tips

- Minimize irrelevant tokens first (reduce noise)
- Choose a good grid / discretization

  - Too coarse: large votes obtained when too many different lines correspond to a single bucket
  - Too fine: miss lines because points that are not exactly collinear cast votes for different buckets

- Vote for neighbors (smoothing in accumulator array)
- Use direction of edge to reduce parameters by 1
- To read back which points voted for “winning” peaks, keep tags on the votes
Hough transform for circles

- A circle with radius $r$ and center $(a, b)$ can be described as:

\[ x = a + r \cos(\theta) \]
\[ y = b + r \sin(\theta) \]
Hough transform for circles

- Circle: center \((a, b)\) and radius \(r\)
  \[
  (x_i - a)^2 + (y_i - b)^2 = r^2
  \]

- For a fixed radius \(r\), unknown gradient direction
Hough transform for circles

- Circle: center \((a, b)\) and radius \(r\)

\[
(x_i - a)^2 + (y_i - b)^2 = r^2
\]

- For a fixed radius \(r\), unknown gradient direction

![Image space](image1.png)

![Hough space](image2.png)

Intersection: most votes for center occur here.
Hough transform for circles

For every edge pixel \((x,y)\):

For each possible radius value \(r\):

For each possible gradient direction \(\theta\):

\[
\begin{align*}
// & \text{ or use estimated gradient at } (x,y) \\
a &= x - r \cos(\theta) \quad // \text{ column} \\
b &= y - r \sin(\theta) \quad // \text{ row} \\
H[a,b,r] &= H[a,b,r] + 1
\end{align*}
\]

Modified from Kristen Grauman
Example: detecting circles with Hough

Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).
Example: detecting circles with Hough

Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).
Hough transform: pros and cons

Pros

- All points are processed independently, so can cope with occlusion, gaps
- Some robustness to noise: noise points *unlikely* to contribute *consistently* to any single bin
- Can detect multiple instances of a model in a single pass

Cons

- Complexity of search time for maxima increases exponentially with the number of model parameters
  - If 3 parameters and 10 choices for each, search is $O(10^3)$
- Quantization: can be tricky to pick a good grid size

Adapted from Kristen Grauman
Generalized Hough transform

• We want to find a template defined by its reference point (center) and several distinct types of landmark points in stable spatial configuration

Template

Triangle, circle, diamond: some type of visual token, e.g. feature or edge point

Adapted from Svetlana Lazebnik
Generalized Hough transform

Intuition:

Now suppose those colors encode gradient directions…

Adapted from Kristen Grauman
Define a model shape by its boundary points and a reference point.

**Offline procedure:**

At each boundary point, compute displacement vector: \( r = a - p_i \).

Store these vectors in a table indexed by gradient orientation \( \theta \).
Generalized Hough transform

Detection procedure:
For each edge point:
• Use its gradient orientation $\theta$ to index into stored table
• Use retrieved $r$ vectors to vote for reference point

Assuming translation is the only transformation here, i.e., orientation and scale are fixed.
Generalized Hough transform

- Template representation
  - For each type of landmark point, store all possible displacement vectors towards the center

Template

Model

Svetlana Lazebnik
Generalized Hough transform

• Detecting the template
  • For each feature in a new image, look up that feature type in the model and vote for the possible center locations associated with that type in the model

Test image

Model

Svetlana Lazebnik
Application: Hough for object detection

• Index displacements by “visual codeword”

B. Leibe, A. Leonardis, and B. Schiele, *Combined Object Categorization and Segmentation with an Implicit Shape Model*, ECCV Workshop on Statistical Learning in Computer Vision 2004
RANSAC

- RANdom Sample Consensus

- **Approach**: we want to avoid the impact of outliers, so let’s look for “inliers”, and use those only.

- **Intuition**: if an outlier is chosen to compute the current fit, then the resulting line won’t have much support from rest of the points.

Kristen Grauman
RANSAC: General form

• **RANSAC loop:**
  1. Randomly select a *seed group* of $s$ points on which to base model estimate (e.g. $s=2$ for a line)
  2. Fit model to these $s$ points
  3. Find *inliers* to this model (i.e., points whose distance from the line is less than $t$)
  4. Repeat $N$ times

• Keep the model with the largest number of inliers

Adapted from Kristen Grauman and Svetlana Lazebnik
RANSAC

(\textit{RANdom SAmple Consensus})

Fischler & Bolles in ‘81.

Line fitting example

Algorithm:

1. \textbf{Sample} (randomly) the number of points required to fit the model
2. \textbf{Solve} for model parameters using samples
3. \textbf{Score} by the fraction of inliers within a preset threshold of the model

\textbf{Repeat} 1-3 until the best model is found with high confidence
RANSAC

(RANdom SAmple Consensus):

Fischler & Bolles in ‘81.

Line fitting example

Algorithm:

1. **Sample** (randomly) the number of points required to fit the model ($\# = 2$)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

**Repeat** 1-3 until the best model is found with high confidence
**Algorithm:**

1. **Sample** (randomly) the number of points required to fit the model ($\#=2$)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

**Repeat** 1-3 until the best model is found with high confidence

*Fischler & Bolles in ‘81.*

*Line fitting example*
RANSAC

(RANdom SAmple Consensus) :

Fischler & Bolles in ‘81.

Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence

Line fitting example

\[ N_I = 6 \]
RANSAC

(RANdom SAmple Consensus) :

Fischler & Bolles in ‘81.

Line fitting example

Algorithm:

1. **Sample** (randomly) the number of points required to fit the model (#=2)
2. **Solve** for model parameters using samples
3. **Score** by the fraction of inliers within a preset threshold of the model

Repeat 1-3 until the best model is found with high confidence
How to choose parameters?

- **Number of samples** $N$
  - Choose $N$ so that, with probability $p$, at least one random sample is free from outliers (e.g. $p=0.99$) (outlier ratio: $e$)

- **Number of sampled points** $s$
  - Minimum number needed to fit the model

- **Distance threshold** $\delta$
  - Choose $\delta$ so that a good point with noise is likely (e.g., prob=0.95) within threshold
  - E.g. for zero-mean Gaussian noise with std. dev. $\sigma$: $\delta^2 = 3.84\sigma^2$

$$N = \frac{\log(1 - p)}{\log(1 - (1 - e)^s)}$$

Explanation in Szeliski 6.1.4

<table>
<thead>
<tr>
<th>$s$</th>
<th>5%</th>
<th>10%</th>
<th>20%</th>
<th>25%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>11</td>
<td>17</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td>19</td>
<td>35</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>13</td>
<td>17</td>
<td>34</td>
<td>72</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>6</td>
<td>12</td>
<td>17</td>
<td>26</td>
<td>57</td>
<td>146</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>7</td>
<td>16</td>
<td>24</td>
<td>37</td>
<td>97</td>
<td>293</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>8</td>
<td>20</td>
<td>33</td>
<td>54</td>
<td>163</td>
<td>588</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>9</td>
<td>26</td>
<td>44</td>
<td>78</td>
<td>272</td>
<td>1177</td>
</tr>
</tbody>
</table>
RANSAC pros and cons

• Pros
  • Applicable to many different problems, e.g. image stitching, relating two views
  • Often works well in practice

• Cons
  • Lots of parameters to tune (see previous slide)
  • Doesn’t work well for low inlier ratios (too many iterations, or can fail completely)

Adapted from Svetlana Lazebnik
Plan for this lecture

• Group pixels into:
  – Edges: Extract gradients and threshold
  – Lines: Find which edge points are collinear or belong to another shape
  – Segments: Find which pixels form a consistent region, e.g. via clustering

• Transform pixels:
  – Find relationships between multiple views of the same world point
  – Both parts rely on finding geometric relationships between pixels
Edges vs Segments

- Edges: More low-level; don’t need to be closed
- Segments: Ideally one segment for each semantic group/object; should include closed contours

Figure adapted from J. Hays
• These intensities define the three groups.
• We could label every pixel in the image according to which of these primary intensities it is.
  • i.e., segment the image based on the intensity feature.
• What if the image isn’t quite so simple?

Source: K. Grauman
Now how to determine the three main intensities that define our groups?

- We need to *cluster*.
• Goal: choose three “centers” as the representative intensities, and label every pixel according to which of these centers it is nearest to.

• Best cluster centers are those that minimize \(\text{sum of squared differences} \) (SSD) between all points and their nearest cluster center \(c_i\):

\[
\sum_{\text{clusters } i} \sum_{\text{points } p \text{ in cluster } i} \| p - c_i \|^2
\]

Source: K. Grauman
Clustering

• With this objective, it is a “chicken and egg” problem:
  – If we knew the **cluster centers**, we could allocate points to groups by assigning each to its closest center.
  – If we knew the **group memberships**, we could get the centers by computing the mean per group.

Source: K. Grauman
K-means clustering

- Basic idea: randomly initialize the $k$ cluster centers, and iterate between the two steps we just saw.

1. Randomly initialize the cluster centers, $c_1, \ldots, c_k$
2. Given cluster centers, determine points in each cluster
   - For each point $p$, find the closest $c_i$. Put $p$ into cluster $i$
3. Given points in each cluster, solve for $c_i$
   - Set $c_i$ to be the mean of points in cluster $i$
4. If $c_i$ have changed, repeat Step 2

Properties
- Will always converge to some solution
- Can be a “local minimum” of objective:
  \[
  \sum_{\text{clusters } i} \sum_{\text{points } p \text{ in cluster } i} \|p - c_i\|^2
  \]
K-means

1. Ask user how many clusters they'd like. 
   \( (e.g. k=5) \)
K-means

1. Ask user how many clusters they’d like. *(e.g. k=5)*

2. Randomly guess k center locations
K-means

1. Ask user how many clusters they’d like. \((\text{e.g. } k=5)\)
2. Randomly guess \(k\) cluster Center locations
3. Each datapoint finds out which Center it’s closest to. (Thus each Center “owns” a set of datapoints)
K-means

1. Ask user how many clusters they’d like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it’s closest to.
4. Each Center finds the centroid of the points it owns

Source: A. Moore
**K-means**

1. Ask user how many clusters they’d like. *(e.g. k=5)*

2. Randomly guess k cluster Center locations

3. Each datapoint finds out which Center it’s closest to.

4. Each Center finds the centroid of the points it owns...

5. ...and jumps there

6. ...Repeat until terminated!

Source: A. Moore
K-means converges to a local minimum

How can I try to fix this problem?

Adapted from James Hays
K-means: pros and cons

Pro
• Simple, fast to compute
• Converges to local minimum of within-cluster squared error

Cons/Issues
• Setting k?
  - One way: silhouette coefficient
• Sensitive to initial centers
  - Use heuristics or output of another method
  - Try different initializations
• Sensitive to outliers
• Detects spherical clusters

ALTERNATIVES?

Adapted from K. Grauman
Mean shift algorithm

- The mean shift algorithm seeks *modes* or local maxima of density in the feature space.

Source: K. Grauman
Kernel density estimation

Kernel

Estimated density

Data (1-D)

Source: D. Hoiem
Mean shift

Search window

Center of mass

Mean Shift vector

Slide by Y. Ukrainitz & B. Sarel
Mean shift
Mean shift

Search window
Center of mass

Mean Shift vector

Slide by Y. Ukrainitz & B. Sarel
Mean shift

Search window

Center of mass

Mean Shift vector

Slide by Y. Ukrainitz & B. Sarel
Mean shift

- Search window
- Center of mass
- Mean Shift vector

Slide by Y. Ukrainitz & B. Sarel
Mean shift
Mean shift
Computing the mean shift

Simple Mean Shift procedure:
• Compute mean shift vector

• Translate the Kernel window by $m(x)$

\[
m(x) = \frac{\sum_{i=1}^{n} x_i K(x_i - x)}{\sum_{i=1}^{n} K(x_i - x)} - x
\]

\[
K(x_i - x) = e^{-\frac{||x_i - x||^2}{\sigma}}
\]

Adapted from Y. Ukrainitz & B. Sarel
Points in same cluster converge
Mean shift clustering

- Cluster: all data points in the attraction basin of a mode
- Attraction basin: the region for which all trajectories lead to the same mode
Mean shift clustering/segmentation

- Compute features for each point (color, texture, etc)
- Initialize windows at individual feature points
- Perform mean shift for each window until convergence
- Merge windows that end up near the same “peak” or mode

Source: D. Hoiem
Mean shift segmentation results

http://www.caip.rutgers.edu/~comanici/MSPAMI/msPamiResults.html
Segmentation as clustering

Depending on what we choose as the feature space, we can group pixels in different ways.

Grouping pixels based on intensity similarity

Feature space: intensity value (1-d)

Source: K. Grauman
Adapted from K. Grauman
Segmentation as clustering

Depending on what we choose as the *feature space*, we can group pixels in different ways.

Grouping pixels based on *intensity* similarity

Clusters based on intensity similarity don’t have to be spatially coherent.

Source: K. Grauman
Segmentation as clustering

Depending on what we choose as the feature space, we can group pixels in different ways.

Grouping pixels based on intensity + position similarity

Both regions are black, but if we also include position \((x,y)\), then we could group the two into distinct segments; way to encode both similarity & proximity.

Source: K. Grauman
Segmentation as clustering

Depending on what we choose as the feature space, we can group pixels in different ways.

Grouping pixels based on texture similarity

Feature space: filter bank responses (e.g., 24-d)

Source: K. Grauman
Summary

• Edges: threshold gradient magnitude
• Lines: edge points vote for parameters of line, circle, etc. (works for general objects)
• Segments: use clustering (e.g. K-means) to group pixels by intensity, texture, etc.
Plan for this lecture

• Group pixels into:
  – Edges: Extract gradients and threshold
  – Lines: Find which edge points are collinear or belong to another shape
  – Segments: Find which pixels form a consistent region, e.g. via clustering

• Transform pixels:
  – Find relationships between multiple views of the same world point

  – Both parts rely on finding geometric relationships between pixels
Why multiple views?

- Structure and depth are inherently ambiguous from single views.

- Multiple views help us to perceive 3D shape and depth.

Kristen Grauman, images from Svetlana Lazebnik
Alignment problem

- We previously discussed how to match features across images, of the same or different objects.
- Now let’s focus on the case of “two images of the same object” (e.g. $x_i$ and $x_i'$).
- What transformation relates $x_i$ and $x_i'$?
- In alignment, we will fit the parameters of some transformation according to a set of matching feature pairs (“correspondences”).

Adapted from Kristen Grauman and Derek Hoiem
Motivation: Image mosaics
First, what are the correspondences?

- Compare content in local patches, find best matches.
  - Scan $x_i'$ with template formed from a point in $x_i$, and compute e.g. Euclidean distance between SIFT features of the patches.
Second, what are the transformations?

Examples of transformations:

- translate
- rotate
- change aspect ratio
- squish/shear
- change perspective

Adapted from Alyosha Efros
Parametric (global) warping

Transformation $T$ is a coordinate-changing machine:

$$p' = T(p)$$

What does it mean that $T$ is **global**?

- It is the same for any point $p$
- It can be described by just a few numbers (parameters)

Let's represent $T$ as a matrix:

$$p' = Mp$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix}$$
Scaling a coordinate means multiplying each of its components by a scalar. Uniform scaling means this scalar is the same for all components.

Adapted from Alyosha Efros
Scaling

*Non-uniform scaling*: different scalars per component

Adapted from Alyosha Efros
Scaling

Scaling operation: 

\[ x' = ax \]
\[ y' = by \]

Or, in matrix form:

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix}
= \begin{bmatrix}
  a & 0 \\
  0 & b
\end{bmatrix}
\begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

scaling matrix \( S \)

Adapted from Alyosha Efros
2D Linear transformations

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
\]

Only linear 2D transformations can be represented with a 2x2 matrix.

Linear transformations are combinations of ...

- Scale,
- Rotation,
- Shear, and
- Mirror
What transforms can we write with 2x2 matrix?

2D Scaling?

\[ x' = s_x \cdot x \]
\[ y' = s_y \cdot y \]

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} =
\begin{bmatrix}
s_x & 0 \\
0 & s_y
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
\]

2D Rotate around (0,0)? (see hidden slide)

\[ x' = \cos \Theta \cdot x - \sin \Theta \cdot y \]
\[ y' = \sin \Theta \cdot x + \cos \Theta \cdot y \]

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} =
\begin{bmatrix}
\cos \Theta & -\sin \Theta \\
\sin \Theta & \cos \Theta
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
\]

2D Shear?

\[ x' = x + sh_x \cdot y \]
\[ y' = sh_y \cdot x + y \]

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} =
\begin{bmatrix}
1 & sh_x \\
sh_y & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
\]

Modified from Alyosha Efros

Fig. from https://www.siggraph.org/education/materials/HyperGraph/modeling/mod_tran/2dshear.htm
What transforms can we write w/ 2x2 matrix?

2D Mirror about Y axis?

\[ x' = -x \]
\[ y' = y \]

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix}
= \begin{bmatrix}
  -1 & 0 \\
  0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

2D Mirror over (0,0)?

\[ x' = -x \]
\[ y' = -y \]

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix}
= \begin{bmatrix}
  -1 & 0 \\
  0 & -1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

2D Translation?

\[ x' = x + t_x \]
\[ y' = y + t_y \]

CAN’T DO!
Homogeneous coordinates

To convert to homogeneous coordinates:

\[(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}\]

homogeneous image coordinates

Converting *from* homogeneous coordinates

\[
\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)
\]
Translation

Homogeneous Coordinates

\[
\begin{bmatrix}
\tilde{x}' \\
\tilde{y}' \\
1
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & t_x \\
0 & 1 & t_y \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix} =
\begin{bmatrix}
x + t \cdot t_x \\
y + t \cdot t_y \\
1
\end{bmatrix}
\]

Adapted from Alyosha Efros
2D affine transformations

\[
\begin{bmatrix}
  x' \\
  y' \\
  w'
\end{bmatrix} = \begin{bmatrix}
  a & b & c \\
  d & e & f \\
  0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  w
\end{bmatrix}
\]

Affine transformations are combinations of …

- Linear transformations, and
- Translations

Maps lines to lines, parallel lines remain parallel

[Diagram showing affine transformation]

Adapted from Alyosha Efros
Detour: Keypoint matching for search

1. Find a set of distinctive keypoints
2. Define a region around each keypoint (window)
3. Compute a local descriptor from the region
4. Match descriptors

\[ d(f_A, f_B) < T \]

Adapted from K. Grauman, B. Leibe
Detour: solving for translation with outliers

Given matched points in \{A\} and \{B\}, estimate the translation of the object

\[
\begin{bmatrix}
    x_i^B \\
y_i^B
\end{bmatrix}
= \begin{bmatrix}
    x_i^A \\
y_i^A
\end{bmatrix} + \begin{bmatrix}
t_x \\
t_y
\end{bmatrix}
\]
Detour: solving for translation with outliers

Problem: outliers, multiple objects

Hough transform solution
1. Initialize a grid of parameter values
2. Each matched pair casts a vote for consistent values
3. Find the parameters with the most votes

\[
\begin{bmatrix}
    x_i^B \\
    y_i^B
\end{bmatrix} =
\begin{bmatrix}
    x_i^A \\
    y_i^A
\end{bmatrix} +
\begin{bmatrix}
    t_x \\
    t_y
\end{bmatrix}
\]
Projective transformations

\[
\begin{bmatrix}
  x' \\
  y' \\
  w'
\end{bmatrix} = \begin{bmatrix}
  a & b & c \\
  d & e & f \\
  g & h & i
\end{bmatrix} \begin{bmatrix}
  x \\
  y \\
  w
\end{bmatrix}
\]

Projective transformations:

- Affine transformations, and
- Projective warps

Parallel lines do not necessarily remain parallel
Obtain a wider angle view by combining multiple images.
Image mosaics: Camera setup

Two images with camera rotation but no translation

Adapted from Derek Hoiem
The mosaic has a natural interpretation in 3D

- The images are reprojected onto a common plane
- The mosaic is formed on this plane
- Mosaic is a *synthetic wide-angle camera*
How to stitch together panorama (mosaic)?

Basic Procedure

- Take a sequence of images from the same position
  - Rotate the camera about its optical center
- Compute the homography (transformation) between first and second image
- Transform the second image to overlap with the first (draw first image onto second canvas)
- Blend the two together to create a mosaic
- (If there are more images, repeat)

Adapted from Steve Seitz
To compute the homography given pairs of corresponding points in the images, we need to set up an equation where the parameters of $H$ are the unknowns...
Computing the homography

- Assume we have four matched points: How do we compute homography $H$?

\[
p' = Hp
\]
\[
p' = \begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix}
\]
\[
H = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}
\]
\[
p = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}
\]
\[
h = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix}
\]

Can set scale factor $h_9 = 1$. So, there are 8 unknowns. Need at least 8 eqs, but the more the better…

\[
A = \begin{bmatrix} -x & -y & -1 & 0 & 0 & 0 & xx' & yy' & x' \\ 0 & 0 & 0 & -x & -y & -1 & xy' & yy' & y' \end{bmatrix} h = 0
\]

Adapted from Derek Hoiem, Kristen Grauman

How to stitch together panorama (mosaic)?

Basic Procedure

• Take a sequence of images from the same position
  – Rotate the camera about its optical center

• **Compute the homography** (transformation) between first and second image

• **Transform the second image** to overlap with the first (draw first image onto second canvas)

• Blend the two together to create a mosaic

• (If there are more images, repeat)

Adapted from Steve Seitz
To apply a given homography $H$

- Compute $p' = Hp$ (regular matrix multiply)
- Convert $p'$ from homogeneous to image coordinates

\[
\begin{bmatrix}
w x' \\
w y' \\
w \\
p'
\end{bmatrix} = \begin{bmatrix}
* & * & * \\
* & * & * \\
* & * & * \\
\end{bmatrix} \begin{bmatrix}
x \\
y \\
1 \\
p
\end{bmatrix}
\]
Transforming the second image

**Forward warping:**
Send each pixel \( f(x,y) \) to its corresponding location \( (x',y') = H(x,y) \) in the right image

Modified from Alyosha Efros
Transforming the second image

Forward warping:
Send each pixel $f(x,y)$ to its corresponding location $(x',y') = H(x,y)$ in the right image

Q: what if pixel lands “between” two pixels?
A: distribute color among neighboring pixels $(x',y')$
Next: Stereo vision

- Homography: Same camera center, but camera rotates
- Stereo vision: Camera center is not the same (we have multiple cameras)

- Epipolar geometry
  - Relates cameras from two positions/cameras

- Stereo depth estimation
  - Recover depth from disparities between two images

Adapted from Derek Hoiem
Stereo photography and stereo viewers

Take two pictures of the same subject from two slightly different viewpoints and display so that each eye sees only one of the images.

Invented by Sir Charles Wheatstone, 1838

Image from fisher-price.com

Kristen Grauman
Depth from stereo for computers

Two cameras, simultaneous views

Single moving camera and static scene

Kristen Grauman
Depth from stereo

- Goal: recover depth by finding image coordinate $x'$ that corresponds to $x$, then measuring discrepancy between $x$ and $x'$
Geometry for a simple stereo system

- Assume parallel optical axes, known camera parameters (i.e., calibrated cameras). **What is expression for Z?**

Similar triangles \((p_l, P, p_r)\) and \((O_l, P, O_r)\):

\[
\frac{T + x_l - x_r}{Z - f} = \frac{T}{Z}
\]

Depth is inversely proportional to disparity.

Adapted from Kristen Grauman
Depth from disparity

- We have two images from different cameras.
- First, find **corresponding points** in two images
  - How to do this efficiently?
- Second, **estimate relative depth** from correspondences
Stereo correspondence constraints

- Given $p$ in left image, where can corresponding point $p'$ be?
Epipolar constraint

Geometry of two views constrains where the corresponding pixel for some image point in the first view must occur in the second view.

- It must be on the line where (1) the plane connecting the world point and optical centers, and (2) the image plane, intersect.
- Potential matches for $p$ have to lie on the corresponding line $l'$.
- Potential matches for $p'$ have to lie on the corresponding line $l$.

Adapted from Kristen Grauman, Derek Hoiem
Epipolar geometry: notation

- **Baseline** – line connecting the two camera centers
- **Epipoles**
  = intersections of baseline with image planes
  = projections of the other camera center
- **Epipolar Plane** – plane containing baseline
- **Epipolar Lines** - intersections of epipolar plane with image planes (always come in corresponding pairs)

Adapted from Derek Hoiem
The epipolar constraint is useful because it reduces the correspondence problem to a 1D search along an epipolar line.

See hidden slides for details.
Essential matrix

\[ \mathbf{X}' \cdot (\mathbf{T} \times \mathbf{R}\mathbf{X}) = 0 \]

\[ \mathbf{X}' \cdot ([\mathbf{T} \times] \mathbf{R}\mathbf{X}) = 0 \]

Let \( \mathbf{E} = [\mathbf{T} \times] \mathbf{R} \)

\[ \mathbf{X}' \cdot \mathbf{E}\mathbf{X} = \mathbf{X}'^T \mathbf{E}\mathbf{X} = 0 \]

\( \mathbf{E} \) is called the essential matrix, and it relates corresponding image points between both cameras, given the rotation and translation.

Before we said: If we observe a point in one image, its position in other image is constrained to lie on line defined by above. It turns out that:

- \( \mathbf{E}^T \mathbf{x} \) is the epipolar line \( \mathbf{l}' \) through \( \mathbf{x}' \) in the second image, corresponding to \( \mathbf{x} \).
- \( \mathbf{E}\mathbf{x}' \) is the epipolar line \( \mathbf{l} \) through \( \mathbf{x} \) in the first image, corresponding to \( \mathbf{x}' \).

Adapted from Kristen Grauman, Derek Hoiem
Basic stereo matching algorithm

- For each pixel in the first image
  - Find corresponding epipolar scanline in the right image
  - Search along epipolar line and pick the best match $x'$ (e.g. smallest Euclidean distance between SIFT in patch)
  - Compute disparity $x - x'$ and set $\text{depth}(x) = f*T/(x - x')$

Adapted from Derek Hoiem
Results with window search

Data

Left image

Right image

Predicted depth

Ground truth
Projective structure from motion

- Given: $m$ images of $n$ fixed 3D points
  
  \[ x_{ij} = P_i X_j, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n \]

- Problem: estimate $m$ projection matrices $P_i$ and $n$ 3D points $X_j$ from the $mn$ corresponding 2D points $x_{ij}$
Photo tourism


http://phototour.cs.washington.edu/
3D from multiple images

Summary of multiple views

• Write **2d transformations** as matrix-vector multiplication

• **Fitting transformations:** Solve for unknown parameters given corresponding points from two views – linear, affine, projective (homography)

• **Mosaics:** Uses homography and image warping to merge views taken from same center of projection

• **Stereo depth estimation:** Find corresponding points along epipolar scanline, then measure disparity (as inverse to depth)

• **Epipolar geometry:** Matching point in second image is on a line passing through its epipole; makes search for correspondences quicker

Adapted from Kristen Grauman and Derek Hoiem